PREDICTION OF CAB FARE AMOUNT

Venkata Sai Maneesha Gara

25 June 2019

Table of Contents

List of figures

Introduction

1. Problem Statement

The objective of this case is predication of cab fare amount for each trip based on factors that are effect the fare amount like season , time , distance , passengercount. The details of data attributes in the dataset are as follows –

Variable	description
pickup_datetime	timestamp value indicating when the cab ride started.
pickup_longitude	float for longitude coordinate of where the cab ride started.
pickup_latitude	float for latitude coordinate of where the cab ride started.
dropoff_longitude	float for longitude coordinate of where the cab ride ended.
dropoff_latitude	float for latitude coordinate of where the cab ride ended.
passenger_count	an integer indicating the number of passengers in the cab ride.
Fare_amount (target variable)	an integer indicating the amount of fare calculated for each trip

1.1. actual features

2. Sample Data set

Below is the structure of the data. 16067 obesrvation and 7 columns with object and float64 as a datatype.

```
In [107]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16067 entries, 0 to 16066
Data columns (total 7 columns):
fare_amount          16043 non-null object
pickup_datetime      16067 non-null object
pickup_longitude     16067 non-null float64
pickup_latitude      16067 non-null float64
dropoff_longitude    16067 non-null float64
dropoff_latitude     16067 non-null float64
passenger_count      16012 non-null float64
dtypes: float64(5), object(2)
memory usage: 878.7+ KB
```

Fig1.1. structure of sample data

In [108]: `train.head()`

Out[108]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

Fig1.2. first 5 observations

Data Preprocessing and Exploratory Data Analysis

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis.

1. TypeConversion

To perform any techniques , the data should be in relavant datatype.hence we need to convert the data type as per the requirement.

The follow are the major conversion we need to do –

- 'pickup_latitude','pickup_longitude','dropoff_longitude','dropoff_latitude'columns should be float type check and convert
- passenger_count should be an integer value
- fare_amount should be an integer value and should be round off to 2 digits

2. Missing Value Analysis

To perform a missing value we have different statistical methods like mean , mode , median and advance data mining methods like KNN imputation. Check for the best approach to fillup the missing values. if you fillup without considering fare_amount and distance and also passenger count,the model will not be accurate hence it is safer to dropoff those columns instead of inserting it with some random values.in the train dataset provided we have missing values in fare_amount and passengers_count so totally 79 rows will be dropped off.

```
missing values in train passenger_count      55
fare_amount            24
dropoff_latitude        0
dropoff_longitude       0
pickup_latitude         0
pickup_longitude        0
pickup_datetime         0
dtype: int64
missing value in test passenger_count      0
dropoff_latitude        0
dropoff_longitude       0
pickup_latitude         0
pickup_longitude        0
pickup_datetime         0
dtype: int64
```

Fig2.1. missing value analysis

3. Outlier Analysis

An outliers are the datapoints that deviates significantly from other observations. The dataset outliers can be identified by validating data upon specific range. The following is the approach followed to identify outliers and remove it. Donot replace with any random values as it misguide our model to give wrong results.

- **Latitude , longitude** values can be corrected based on the min , max values of the New york City co-ordinates. On referring to google identified that below are the boundaries of newyork city.
  - lat_min=37
  - lat_max=45.0153
  - lon_min=-79.7624
  - lon_max=-71.7517
- **Passenger_count** , in New York city for a trip at max there can be only 6 passenger in cab and there is no meaning in calculating fare_amount where there are no passengers. hence the passengers_count should be in the range of 1 to 6.
- **fare_amount** in train dataset , in New York city the basic fare of a cab is $2.5 and at max it can be $250 in usual scenarios. hence the fare_amount should be in range of 2.5 to 250 and also some of the data might be string format , in that case they should be remove on validating with regular expressions.
- **Pickup_datetime ,** the values might not be in the standard timestamp format , hence with help of regular expression validation we can check and remove the outliers.

Remove all the outliers from the training dataset before you proceed to feature engineering

4. Feature Engineering

In this stage , we will derive more features from the existing features which can add value to our model.The follow are the list of features that can be derived –

Derived Variable	description
Distance	Float value calculated based on the latitude and longitude values given for pickup and drop locations.
Date	Date, derived from the pickup_datetime
Day_of_week	Integer , derived from the pickup_datetime
Month	Integer , derived from the pickup_datetime
year	Integer, derived from the pickup_datetime
Day	Integer, derived from the pickup_datetime

1.2 .Table of derived variables

- The **Distance** between pickup location and drop location can be calculated using harverine Distance Formula where it calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as-the-crow-flies' distance between the points. Note the values of latitude and logitude must be in radian format.

$$\varphi = d \,.\, pi / 180 \quad \text{\# where pi = 3.14 and d is the degree of latitude or longitude.}$$

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km);*

- **Date , Day_of_week , Month,year,Day** can be derived , provided if the pickupDateTime is in correct format.

The structure and statistics of train data set after feature engineering , outlier analysis and missing value analysis.

Out[232]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hr	month
count	15696.000000	15696.000000	15696.000000	15696.000000	15696.000000	15696.000000	15696.000000	15696.000000	15696.000000
mean	11.298077	-73.974835	40.750901	-73.973863	40.750901	1.645005	5.170808	16.770387	5.170808
std	9.597119	0.041506	0.037971	0.039350	0.037971	1.262036	2.722191	6.401340	2.722191
min	2.500000	-74.438233	39.603178	-74.429332	39.603178	1.000000	1.000000	0.000000	1.000000
25%	6.000000	-73.992394	40.736588	-73.991373	40.736588	1.000000	4.000000	13.000000	4.000000
50%	8.500000	-73.982067	40.753302	-73.980575	40.753302	1.000000	4.000000	20.000000	4.000000
75%	12.500000	-73.968110	40.767801	-73.965460	40.767801	2.000000	6.000000	20.000000	6.000000
max	180.000000	-73.137393	41.366138	-73.137393	41.366138	6.000000	12.000000	23.000000	12.000000

Fig4.1. Summary of data after Feature Engineering

5. Visualizations , Feature Scaling and Selection

In this stage , we select features that are required in model we build. Before filtering feature the following are the analysis we need to do

- Evaluate the relation between features with help of visualization techniques
- Cacluate the correlation between the predicators(dependant variables)
- Analyse and filter required features to avoid multi-collinearity problem

a. Relation beween fare_amount vs passenger_count per year

The below visualization is about the average fare_amount and based on the year and also passenger count.

- the fare_amount is between $10 to $15 on an average with passengers 1 or 2 is high.
- The average fare_amount got increased gradually year to year and it is maximum in 2014.

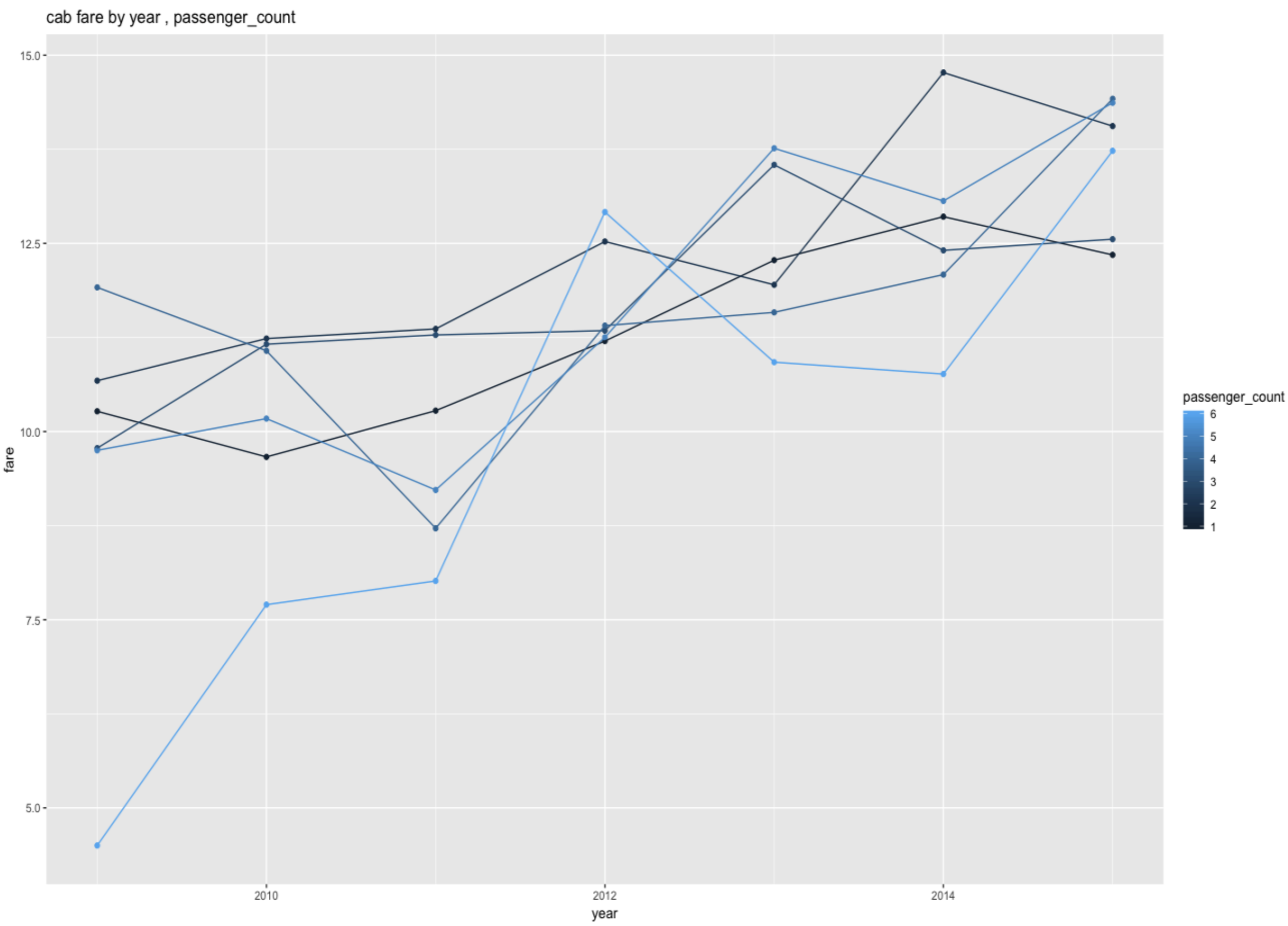

Fig5.1. Average Fare_Amount & passenger count per year

b. Distribution of Trips Fare

below visualization shows the logarithmic distribution of fare amount. It shows that the fare_amount is normally distributed. hence feature scaling is not required.

```
Out[234]: Text(0.5, 1.0, 'Distribution of Trip Fare')
```

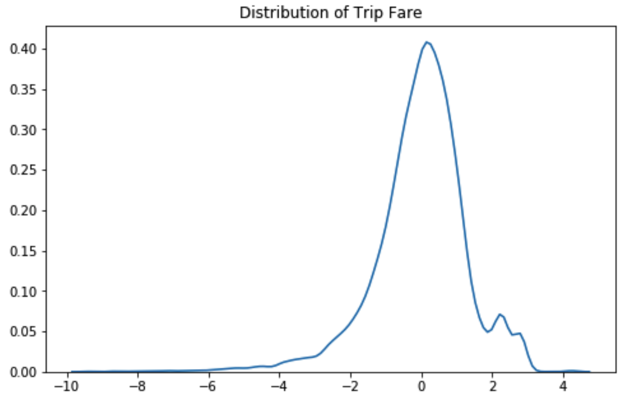

Fig5.2.Distribution of Trips Fare

c. Distribution of Trips Distance

below visualization shows the logarithmic distribution trip distance .It shows that the fare_amount is normally distributed. hence feature scaling is not required.

```
Out[201]: Text(0.5, 1.0, 'Distribution of Trip Distance (log scale)')
```

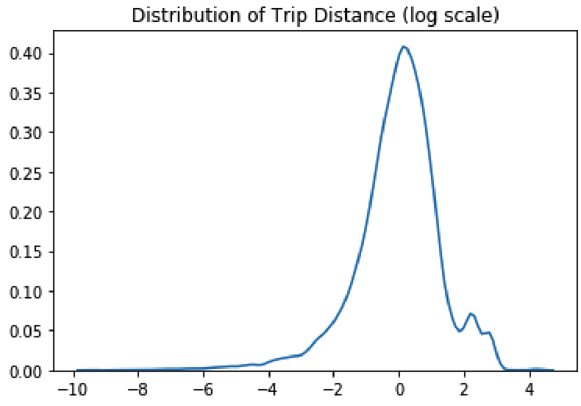

Fig5.3.Distribution of Trips Distance

d. Distribution of pickup location

The following visualization states that pickup location is mostly in between 40.7 to 40.8 of latitude and -74 and -73.95 of longitude respectively.

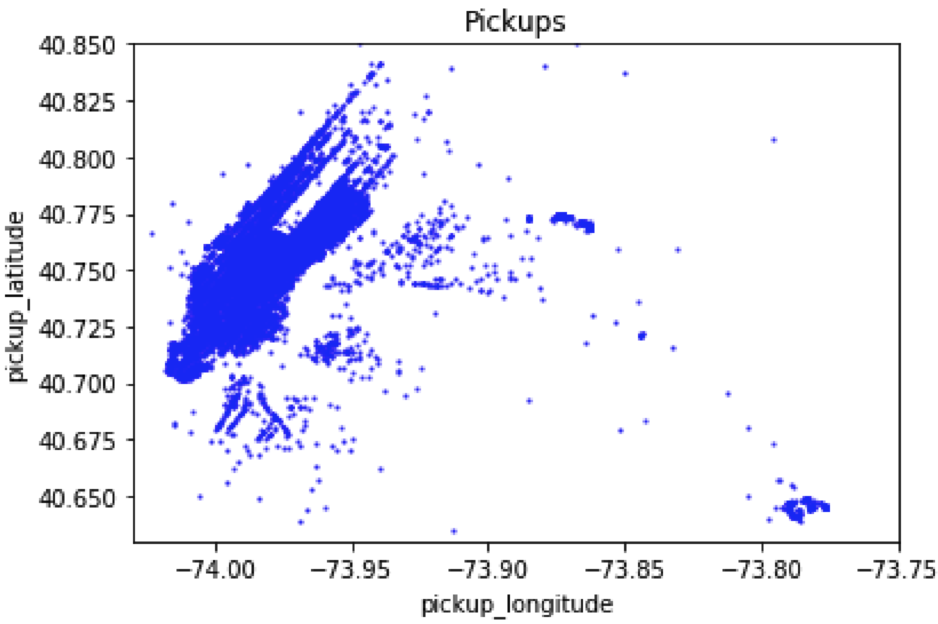

Fig5.4. Distribution of Pickup Locations

e. Distribution of drop location

The following visualization states that drop location is mostly in between 40.7 to 40.8 of latitude and -74 and -73.95 of longitude respectively. We can come up to conclusion in this locations even though the cab is done with one trip. There would be chances of another pickup.

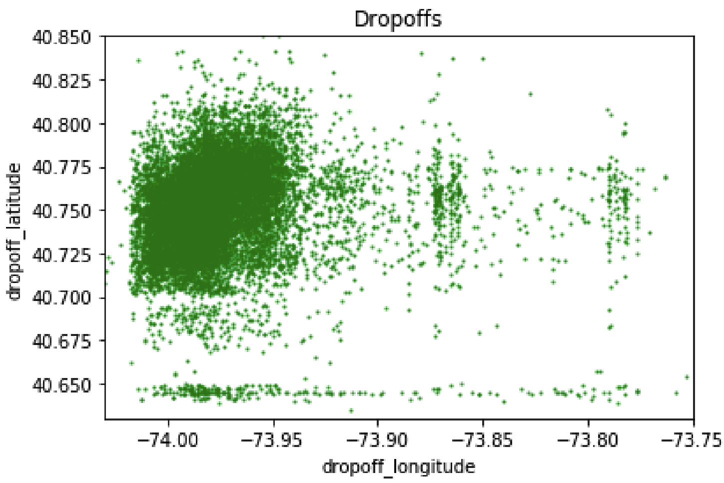

Fig5.5.Distribution of Drop Locations

f. Average trip rate on weekly basis

The below visualization will give us the average trips takens on weekly basis. Compared to all the days. Saturday are have more trips.

```
Out[240]: Text(0.5, 1.0, 'Avg  trip rate on weekly basis')
```

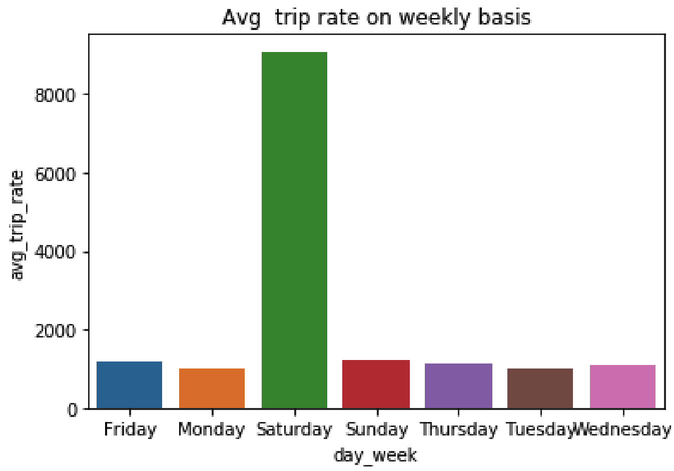

Fig5.6. Triprate Analysis on weekly basis

g. Average fare amount on monthly basis

below visualization will tell you the fare_amount distribution over months. here we could see that there no much difference.

```
Out[368]: Text(0.5, 1.0, 'Avg Fare Amount over month||')
```

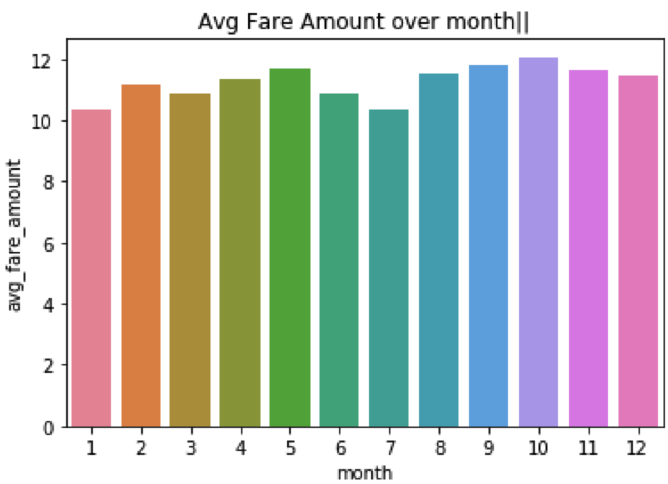

Fig5.7.Average fare amount on monthly basis

h. Correlation Analysis

From below visualization,there is no corelation between the features from below graph , except for 2 set of variables that is the (pickup latitude, dropoff latitude) and (day,month) , but we cannot remove these columns as they are highly dependent to draw conclusions on the fareamount based on the day and latitude.

```
Out[251]: <matplotlib.axes._subplots.AxesSubplot at 0x1c354b4438>
```

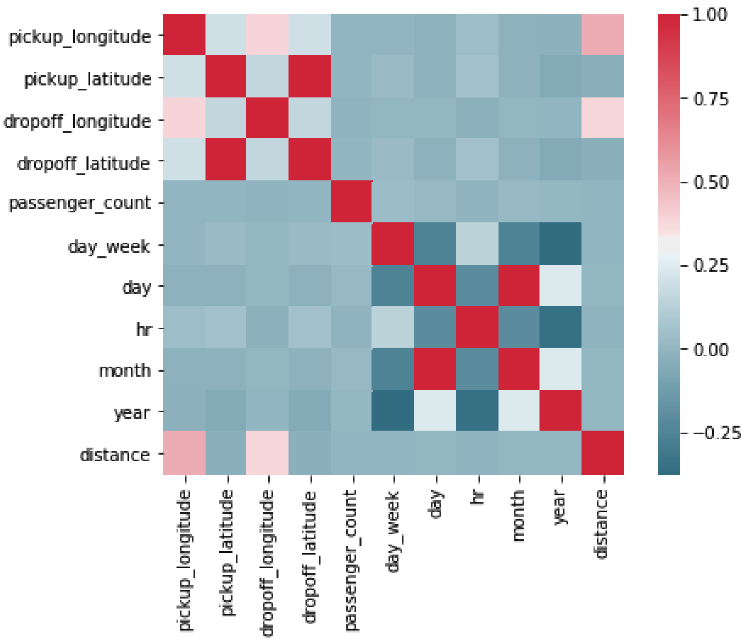

Fig5.8.Correlation Analysis

Conclusions out of feature engineering

- **distance,pickup_latitude,pickup_longitude,drop_off_latitude,drop_off_longitude,passenger_count** are correlated with the traget variable **“fare_amount”**
- **Date , pickup_date** columns can be dropped off as they are substituded by **day month,year,hr.**
- convert **day_of_the_week** to numerical variable.
- The date features **day,hr,month,year,day_of_week** will be retained.

hence after feature engineering , the shape of our training dataset is (15696, 12) and of testdata set is (9914, 11) observations and features respectivetly.

Model Development

1. Manual Calculation of Fare Amount

in this model , we first calculate fare rate per km. Calculate the overall fareamount which distance is provided in kms.The formula to calculate rate per km is

fare_amount = distance * 1.76+2.5

rate per km = average of fare_amount/average of distance

it may vary based on other factor time and the area of travel.The charges will be high if the trip is to/from airport.

2. Multi Linear Regression Model

The target variable – fare_amount is a continuous variable hence we will go with the regression analysis.

- The training data available will be divided into 2 groups test and train.
- The train dataset will be further divided into 2 groups before we apply our model, one with set of predicators and one with target variable i.e., X_train,Y_train.
- Consider X_train,Y_train datasets, passed them as arguments to linear regeression model.
- MultiLinear Regression works based on below statistical formula –

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon$$

- Predict the values of fare_amount after model is build by passing X_test dataset. This will give the estimated fare_amount.
- Caculate MAPE , MSE , RMSE to compare the actual result with predicted result to identify error.

3. Ordinary Least Square Model (OLS)

Ordinary least squares (OLS) regression is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable; the method estimates the relationship by minimizing the sum of the squares in the difference between the observed and predicted values of the dependent variable configured as a straight line.

The process of applying this model is similar to Multi Linear Regression. upon applying OLS model. we need to analyse the data behaviour based on the summary.

```
Test RMSE after applying  OLS  : 0.147
                            OLS Regression Results
==============================================================================
Dep. Variable:            fare_amount   R-squared:                       0.801
Model:                            OLS   Adj. R-squared:                  0.800
Method:                 Least Squares   F-statistic:                     4896.
Date:                Wed, 26 Jun 2019   Prob (F-statistic):               0.00
Time:                        06:23:53   Log-Likelihood:                -36387.
No. Observations:               10987   AIC:                         7.279e+04
Df Residuals:                   10978   BIC:                         7.286e+04
Df Model:                           9
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
pickup_longitude    -20.3507      1.547    -13.156      0.000     -23.383     -17.318
pickup_latitude     -16.4021      0.739    -22.182      0.000     -17.851     -14.953
dropoff_longitude    10.8310      1.565      6.919      0.000       7.762      13.900
dropoff_latitude    -16.4021      0.739    -22.182      0.000     -17.851     -14.953
passenger_count       0.0463      0.051      0.916      0.360      -0.053       0.145
day_week              0.2238      0.041      5.430      0.000       0.143       0.305
day                   0.0204      0.012      1.661      0.097      -0.004       0.045
hr                    0.0293      0.011      2.741      0.006       0.008       0.050
month                 0.0204      0.012      1.661      0.097      -0.004       0.045
year                  0.3172      0.036      8.732      0.000       0.246       0.388
distance              2.2197      0.021    105.487      0.000       2.178       2.261
==============================================================================
Omnibus:                     7452.653   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         10570372.917
Skew:                           1.762   Prob(JB):                         0.00
Kurtosis:                     154.913   Cond. No.                     1.07e+20
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.93e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Fig6.1.OLS analysis

- the major focus will be on R-squared and adjusted-R-squared. we find that the value is around 0.8 which is significant to state that the predicator are highly corelated with target variable.
- There very minor difference in AIC and BIC and aslo R-squared and adjusted R-square which is good result.
- calculate MSE,RMSE and MAPE error for further analysis.

4. Decision Tree Model

A **decision tree model** is a **decision** support tool that uses a **tree**-like graph or model of **decisions** and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. The basic process of applying any model is similar to Multi Linear Regression.we used DecisionTreeRegressor to build model. Calculate Error metrics for further analysis.

5. RandomForest Model

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. Ensembling is nothing but a combination of weak learners (individual trees) to produce a strong learner. The process of applying this model is similar to Multi Linear Regression. Random Forest Regressor to build model .The no. of trees used is 500 and feature imporance is selected as true.. Calculate Error metrics for further analysis.

Model Validation

The model validation can be done by calculating various error metrics. The following are the list of error metrics used in the model.

- MeanAbsolutePercentageError(MAPE)
- MeanSquareError(MSE)
- RootMeanSquareError(RMSE)

i. Mean Absolute Percentage Error (MAPE)

The mean absolute percentage error (MAPE) is a statistical measure of how accurate a forecast system is. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. Where A_t is the actual value and F_t is the forecast value, this is given by:

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Here A is the actual value and F is the predicted value. Where as n is the no.of observations.

i. Mean Square Error (MSE)

Mean squared error is a single value that provides information about the goodness of fit of the regression line. The smaller the MSE value, the better the fit, as smaller values imply smaller magnitudes of error.

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$* \, n$ is the number of data points
$* \, Y_i$ represents observed values
$* \, \hat{Y}_i$ represents predicted values

ii. Root Mean Square Error (MSE)

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

$$\mathrm{RMSE}_{fo} = [\sum_{i=1}^{N}(z_{f_i} - z_{o_i})^2/N]^{1/2}$$

Where:

- Σ = summation ("add up")
- $(z_{f_i} - Z_{oi})$Sup>2 = differences, squared
- N = sample size.

iii. Error Metrics Comparision

Compared to all the RMSE is more appropriate to represent model performance than the MAPE when the error distribution is expected to be Gaussian. In addition, we show that the RMSE satisfies the triangle inequality requirement for a distance metric.The below image give the comparsion of all the error metrics.

```
error_table
```

	ManualCalculation	LR	DecisionTree	RandomForest	OLS
MAPE	44.169000	39.182607	38.223983	38.268799	39.142989
MSE	73.968514	37.790028	31.057557	30.106247	38.030192
RMSE	8.600495	6.147359	5.572931	5.486916	6.166862

Fig6.2.Error Metrics

Conclusion

- Based on the error analysis ,we conclude that on applying Random Forest give us more accurate result than OLS , multilinear , Decisiontree and manual calculations.

- Once the model is fixed we can now predict the fare_amount for the any test data provided. PFB summary of test values predicted.

```
test['fare_amount'].describe()
```

```
count    9914.000000
mean       11.330610
std         7.524217
min         8.263000
25%         8.263000
50%         8.263000
75%        13.563000
max        51.010000
Name: fare_amount, dtype: float64
```

Fig7.1.Predicted Test Fare_amount

- The major features that are influencing fare_amount is the distance ,no. of passenger_count.

APPENDIX

Python-Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import calendar
from math import sin,cos,atan2,sqrt,radians,asin
from datetime import datetime
import calendar
import re
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
###load training and test data and also calculate the missing values############
train = pd.read_csv("train_cab.csv")
test = pd.read_csv("test.csv")

print("missing values in train",train.isnull().sum().sort_values(ascending=False))
print("missing value in test",test.isnull().sum().sort_values(ascending=False))

train.describe()

''' based on the analysis , the latitude and logitude values are matching with the newyork coordinates hence we can
   that the cab fare should be calculated in the newyork city

'''
lat_min=37
lat_max=45.0153
lon_min=-79.7624
lon_max=-71.7517

print(lat_min,',',lat_max)
print(lon_min,',',lon_max)

### methods that are used for data cleaning and exploratory data analysis

def data_cleaning_stage_cooridnates(df):

   #make the co-ordinates which are out of range to null and remove those records later
   df['pickup_latitude']=df.apply(lambda row: np.nan if((row['pickup_latitude'] < lat_min)|(row['pickup_latitude']> lat_max)) else row['pickup_latitude'],axis=1)
   df['dropoff_latitude']=df.apply(lambda row: np.nan if((row['dropoff_latitude'] < lat_min)|(row['dropoff_latitude']> lat_max)) else row['pickup_latitude'],axis=1)
```

```
    df['pickup_longitude']=df.apply(lambda row: np.nan if((row['pickup_longitude'] <
lon_min)|(row['pickup_longitude']> lon_max)) else row['pickup_longitude'],axis=1)
    df['dropoff_longitude']=df.apply(lambda row: np.nan if((row['dropoff_longitude'] <
lon_min)|(row['dropoff_longitude']> lon_max)) else row['dropoff_longitude'],axis=1)

    #function to calculate distance bewteen 2 cooridinates
    def calculateDistance(lat1,long1,lat2,long2):
        radius = 6371
        dlat = np.abs(np.radians(lat1)-np.radians(lat2))
        dlong = np.abs(np.radians(long1)-np.radians(long2))
        t1 =
(np.sin(dlat/2)**2)+(np.cos(radians(lat1))*np.cos(radians(lat2))*np.sin(dlong/2)**2)
        t2 = 2*(atan2(np.sqrt(t1),np.sqrt(1-t1)))
        return radius*t2

    #call to distance function to calculate based on the latitude and longitude values
provided
    df['distance']=df.apply(lambda
row:calculateDistance(row['pickup_latitude'],row['pickup_longitude'],row['dropoff_latit
ude'],row['dropoff_longitude']),axis=1)

    print("end of the co_ordinates preprocessing")
    return df


def calculateDateTime(df):
    mode_dt = df['pickup_datetime'].mode()[0]
    print(mode_dt)

    #replace the date string with mode if the date doesnot match
    def date_validation(str1):
        r = re.compile('[1-2][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9] [0-2][0-3]:[0-5][0-9]:[0-
5][0-9] UTC')
        if r.match(str1):
            return str1
        else:
            return mode_dt

    def time_date_outliers(value,x,lower,upper):
        value = int(value)
        if (value < lower|value > upper):
            return int(df[x].mode())
        return int(value)

    df['pickup_datetime']=df.apply(lambda
row:date_validation(str(row['pickup_datetime'])),axis=1)
    df['pickup_datetime']=pd.to_datetime(df['pickup_datetime'],format='%Y-%m-%d
%H:%M:%S UTC')
```

```
    df['day_week']=df['pickup_datetime'].apply(lambda
x:calendar.day_name[x.weekday()])
    df['date']= df['pickup_datetime'].dt.date
    df['day']=df['pickup_datetime'].apply(lambda x:x.day)
    df['hr']=df['pickup_datetime'].apply(lambda x:x.hour)
    df['month']=df['pickup_datetime'].apply(lambda x:x.month)
    df['year']=df['pickup_datetime'].apply(lambda x:x.year)

    #month validation
    df["month"]=df.apply(lambda row: time_date_outliers(row["month"],"month",1,12)
if (row["month"] not in range(0,13)) else row["month"],axis=1)

    #day validation
    df["day"]=df.apply(lambda row: time_date_outliers(row["day"],"day",1,30) if
(row["month"] in [4,6,9,11]) else row["month"],axis=1)
    df["day"]=df.apply(lambda row: time_date_outliers(row["day"],"day",1,31) if
(row["month"] in [1,3,5,7,8,10,12]) else row["month"],axis=1)
    df["day"]=df.apply(lambda row: time_date_outliers(row["day"],"day",1,31) if
(row["month"] == 2) else row["month"],axis=1)

    print("end of date time validation")
    #df=df.drop('pickup_datetime',axis=1)
    return df

def calculateFare(df):
    df['fare_amount']=df['fare_amount'].astype(np.float64)
    fare = train['distance'].max()*1.56+52
    ''' the fare can be at max of 250$ , considering airport as highest fare and on
computation the max fare is 165$'''
    return df[(df['fare_amount'] >= 2.5) & (df['fare_amount'] <= 250)]

#assuming that maximum passengers in cab would be 6 and if it considered as bus it is
12, replace the other values with null
def calculatePassengerCount(df):

    m =
df[(df['passenger_count']>=1)|(df['passenger_count']<=6)]['passenger_count'].mode(
)
    df['passenger_count']=df.apply(lambda row:  m if (row['passenger_count'] not in
range(1,7)) else row['passenger_count'],axis=1)
    return df

#replace the string with npa if it contains any apart from numericals
def string_validation(str1):
        if type(str1)==type('string'):
            r = re.compile('.*[(+*)-]|[A-Z]|[a-b].*')
            if r.match(str1):
                return np.nan
            else:
```

```
        return float(str1)

#convert the set of columns to float
def convert_to_float(df,float_columns):
    for i in float_columns:
        df[i]=df[i].astype(float)
    return df

#identify the missing values and drop off
def missing_values(df):
    for i in df.columns:
        if(df[i].isnull().sum() != 0):
            print(i,":missing values before drop:",df[i].isnull().sum())
            df=df.drop(df[df[i].isnull()].index,axis=0)
            print(i,":missing values after drop:",df[i].isnull().sum())
    return df

#type conversions of train data
float_columns=['pickup_latitude','pickup_longitude','dropoff_longitude','dropoff_latitud
e']
train['fare_amount']=train.apply(lambda
row:string_validation(row['fare_amount']),axis=1)
train=convert_to_float(train,float_columns)

#DateTime validation and EDA of train data

''' if the latitude and logitude value are incorrect that data has no value hence we will be
droping all the rows
that are out of newyork city range. The same with the target variable : fare_amount

'''
train=calculateDateTime(train)
#latitude and longitude validation and also distance calculation
train=data_cleaning_stage_cooridnates(train)
#Passenger count validation
train=calculatePassengerCount(train)
#drop the list of missing values
train=missing_values(train)
#convert the values
train['passenger_count']=train['passenger_count'].astype(int)

train = calculateFare(train)

#DateTime validation and EDA of test data
test=calculateDateTime(test)
#latitude and longitude validation and also distance calculation
test=data_cleaning_stage_cooridnates(test)
#drop the list of missing values
test=missing_values(test)
```

```
#Passenger count validation
test=calculatePassengerCount(test)
#type conversion od test data
test=convert_to_float(test,float_columns)
test['passenger_count']=test['passenger_count'].astype(int)

'''Exploratory data analyis  the major variables are fare_amount , distance , passenger
count '''

plt.figure(figsize=(8,5))
sns.kdeplot(np.log(train['distance'].values)).set_title("Distribution of Trip Fare")

city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)

train.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude',
                color='green',
                s=1.5, alpha=.6)
plt.title("Dropoffs")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)

train.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',
                color='blue',
                s=1.5, alpha=.6)
plt.title("Pickups")

plt.ylim(city_lat_border)
plt.xlim(city_long_border)

#scaling of the distance is good it is normally distributed
sns.kdeplot(np.log(train['distance'].values)).set_title("Distribution of Trip Distance (log
scale)")

trips_year=train.groupby(['year'])['pickup_datetime'].count().reset_index().rename(col
umns={'pickup_datetime':'Num_Trips'})
trips_year.head()
sns.barplot(x='year',y='Num_Trips',data=trips_year)

trips_year_fareamount=train.groupby(['year'])['fare_amount'].mean().reset_index().re
name(columns={'fare_amount':'avg_fare_amount'})
sns.barplot(x='year',y='avg_fare_amount',data=trips_year_fareamount).set_title("Avg
Fare Amount over Years")

train['passenger_count'].value_counts().plot.bar(color = 'b', edgecolor = 'k')
plt.title('Histogram of passenger counts')
plt.xlabel('Passenger counts')
plt.ylabel('Count')
```

```
trips_day_fareamount=train.groupby(['day_week'])['pickup_datetime'].count().reset_in
dex().rename(columns={'pickup_datetime':'avg_trip_rate'})
sns.barplot(x='day_week',y='avg_trip_rate',data=trips_day_fareamount).set_title("Avg
trip rate on weekly basis")

train.groupby('passenger_count').size()

trips_passenger_count=train.groupby(['passenger_count'])['pickup_datetime'].count().
reset_index().rename(columns={'pickup_datetime':'avg_trip_rate'})
sns.barplot(x='passenger_count',y='avg_trip_rate',data=trips_passenger_count).set_titl
e("trips_passenger_count")


train['day_week'] =
train['day_week'].replace(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturda
y','Sunday'],[1,2,3,4,5,6,7])

test['day_week'] =
test['day_week'].replace(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday
','Sunday'],[1,2,3,4,5,6,7])

train.shape


df_corr= train.drop(columns=['fare_amount'])
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = df_corr.corr()

#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
         square=True, ax=ax)

train=train.drop('date',axis=1)
test=test.drop('date',axis=1)


train=train.drop('pickup_datetime',axis=1)
test=test.drop('pickup_datetime',axis=1)

def distance_fare(df):
   df=df.drop(train[(train['H_Distance']==0)&(train['fare_amount']==0)].index, axis =
0)
   return df

def compute_rmse(actual, predicted):
   return np.sqrt(np.mean((actual - predicted)**2))
```

```
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
    return mape

def farerate_rmse(df, rate, name):
    print(name,":RMSE for fare rate is
=",round(compute_rmse(df['fare_amount'],train['distance']*1.76+2.5),3))


def mean_squared_error(actual, predicted):
    return (np.mean((actual - predicted)**2))


rate = round(train['fare_amount'].mean() / train['distance'].mean(),3)

#fare = train['distance']*1.76+2.5

print("Rate = ${0}/km".format(rate))

farerate_rmse(train, rate, 'Train')

print("Train :MAPE for fare rate is =",round(MAPE(train['fare_amount'],rate *
train['distance']),3))
error_table=pd.DataFrame(index=['MAPE','MSE','RMSE'])

error_table['ManualCalculation']=[round(MAPE(train['fare_amount'],train['distance']*
1.76+2.5),3),mean_squared_error(train['fare_amount'], rate *
train['distance']),compute_rmse(train['fare_amount'],rate * train['distance'] )]

####################################LINEAR REGRESSION
###############################################################
#####
from sklearn.linear_model import LinearRegression

Y =train['fare_amount']
X = train.drop(columns=['fare_amount'])

X1_train, X1_test, Y1_train, Y1_test = train_test_split(X, Y, test_size=0.3,
random_state=40)

lr = LinearRegression().fit(X1_train, Y1_train)
predictions_LR = lr.predict(X1_test)


print("Test  MAPE after applying  Linear Regression : %.3f" % MAPE(Y1_test,
predictions_LR))
```

```
print("Test  MSE after applying Linear Regression : %.3f" %
mean_squared_error(Y1_test, predictions_LR))
print("Test RMSE after applying Linear Regression  : %.3f" % compute_rmse(Y1_test,
predictions_LR))

error_table['LR']=[MAPE(Y1_test, predictions_LR),mean_squared_error(Y1_test,
predictions_LR),compute_rmse(Y1_test, predictions_LR)]

#################################### DECISION TREE REGERESSION
##########################################################

from sklearn.tree import DecisionTreeRegressor

X2_train, X2_test, Y2_train, Y2_test = train_test_split(X, Y, test_size=0.3,
random_state=40)

fit_DT = DecisionTreeRegressor(max_depth=2).fit(X2_train,Y2_train)
predictions_DT = fit_DT.predict( X2_test)

print("Test  MAPE after applying  Decision Regression : %.3f" % MAPE(Y2_test,
predictions_DT))
print("Test  MSE after applying Decision Regression : %.3f" %
mean_squared_error(Y2_test, predictions_DT))
print("Test RMSE after applying Decision Regression  : %.3f" % compute_rmse(Y2_test,
predictions_DT))

error_table['DecisionTree']=[MAPE(Y2_test,
predictions_DT),mean_squared_error(Y2_test, predictions_DT),compute_rmse(Y2_test,
predictions_DT)]
################################## RANDOM FOREST
##############################################################
##########
from sklearn.ensemble import RandomForestRegressor

X3_train, X3_test, Y3_train, Y3_test = train_test_split(X, Y, test_size=0.3,
random_state=40)


rf = RandomForestRegressor(max_depth=2, random_state=0,
n_estimators=100).fit(X3_train, Y3_train)
predictions_RF = rf.predict(X3_test)

print("Test  MAPE after applying  Random Forest %.3f" % MAPE(Y3_test,
predictions_RF))
print("Test  MSE after applying  Random Forest %.3f" %  mean_squared_error(Y3_test,
predictions_RF))
print("Test RMSE after applying  Random Forest  : %.3f" % compute_rmse(Y3_test,
predictions_RF))
```

```
error_table['RandomForest']=[MAPE(Y3_test,
predictions_RF),mean_squared_error(Y3_test, predictions_RF),compute_rmse(Y3_test,
predictions_RF)]

################################## OLS MODEL IN LINEAR REGRESSION
########################################################
import statsmodels.api as sm

X4_train, X4_test, Y4_train, Y4_test = train_test_split(X, Y, test_size=0.3,
random_state=40)

model = sm.OLS(Y4_train,X4_train).fit()

# make the predictions by the model
predictions_OLS = model.predict(X4_test)

print("Test  MAPE after applying  OLS %.3f" % MAPE(Y4_test, predictions_LR))
print("Test  MSE after applying  OLS %.3f" %  mean_squared_error(Y4_test,
predictions_LR))
print("Test RMSE after applying  OLS  : %.3f" % compute_rmse(Y4_test,
predictions_LR))

# Print out the statistics
print(model.summary())

error_table['OLS']=[MAPE(Y4_test, predictions_OLS),mean_squared_error(Y4_test,
predictions_OLS),compute_rmse(Y4_test, predictions_OLS )]

error_table

'''based on the analysis we can fix random forest as the accurate model as the error rate
is less'''

############################# PREDICTING THE FARE  AMOUNT of TEST
DATE ###################################################

test['fare_amount']=np.round(rf.predict(test),3)

test.to_csv("test_output.csv",index=False)
```

End of Python Code

R-Code

```r
rm(list = ls())
getwd()
setwd("~/Documents/datascience/cabfareprediction")
####################################load library files
 #################################################################
x = c("geosphere","stringr","DMwR","caret","rpart","MASS","usdm")
lapply(x, require, character.only = TRUE)

################################read train
 data###########################################################
 ##########
train=read.csv("train_cab.csv",header = T,na.strings = c(""," ",NA))
test=read.csv("test.csv",header = T,na.strings = c(""," ",NA))
#####################data preprocessing and EDA methods
 ###############################################################
cat("no. of missing values in train dataset")
colSums(is.na(train))

cat("no. of missing values in test dataset")
colSums(is.na(test))

#################################### type conversions
 ################################################################
cnames = c("pickup_latitude","pickup_longitude","dropoff_latitude","dropoff_longitude")


for(i in 1:ncol(train)){

 if(class(train[,i]) == 'list'){

  train[,i] = as.numeric(train[,i])

 }
}
 train$pickup_datetime = as.character(train$pickup_datetime)
 train$passenger_count = as.integer(as.character(train$passenger_count))


 for(i in 1:ncol(test)){

  if(class(train[,i]) == 'list'){

   test[,i] = as.numeric(test[,i])

  }
 }
 test$pickup_datetime = as.character(test$pickup_datetime)
 test$passenger_count = as.integer(as.character(test$passenger_count))

###################################################################
 ######################################################
```

```
cord = c("pickup_latitude","pickup_longitude","dropoff_latitude","dropoff_longitude")
print("set the range of latitude and longitude for the newyork country")
lat_min=37
lat_max=45.0153
lon_min=-79.7624
lon_max=-71.7517
radius = 6371
cat("latitude limits",lat_min,",",lat_max)
cat("longitude limits",lon_min,",",lon_max)

data_cleaning_stage_cooridnates<-function(df){

 outliers<-function(x,l,r){if(x<l | x > r){return(NA)}else{return(x)}}

 df$pickup_latitude= lapply(df$pickup_latitude,function(x){ outliers(x,lat_min,lat_max)})
 df$dropoff_latitude= lapply(df$dropoff_latitude,function(x){ outliers(x,lat_min,lat_max)})
 df$pickup_longitude= lapply(df$pickup_longitude,function(x){ outliers(x,lon_min,lon_max)})
 df$dropoff_longitude= lapply(df$dropoff_longitude,function(x){ outliers(x,lon_min,lon_max)})

 df=na.omit(df)

 for(i in 1:ncol(df)){

  if(class(df[,i]) == 'list'){

   df[,i] = as.numeric(df[,i])

  }
 }


 radians=function(x){
  x = as.numeric(x)
 return(x*pi/180)}

 distance=function(lat1,lat2,long1,long2)
 {
  dlat = abs(radians(lat1)-radians(lat2))
  dlong = abs(radians(long1)-radians(long2))
  t1 = (sin(dlat/2)**2)+(cos(radians(lat1))*cos(radians(lat2))*sin(dlong/2)**2)
  t2 = 2*(atan2(sqrt(t1),sqrt(1-t1)))
  return(abs(6371*t2))

 }

 for(i in 1:nrow(df)){df$distance[i] =
distance(df$pickup_latitude[i],df$dropoff_latitude[i],df$pickup_longitude[i],df$dropoff_longitu
de[i])}
 print("end of co-ordinates preprocessing")
 return(df)
}

data_cleaning_stage_date_time<-function(df){
   for(i in 1:nrow(df)){
```

```
    x=as.character(df$pickup_datetime[i])
    d = str_detect("2015-01-27 13:08:24 UTC", "[1-2][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9] [0-
 2][0-3]:[0-5][0-9]:[0-5][0-9] UTC")
    if(length(x)==length("2009-06-15 17:26:21 UTC")){
      if(d){
      df$year[i] = as.integer(substring(strsplit(x," "), 4,7))
      df$month[i]=as.integer(substring(strsplit(x," "), 9,10))
      df$day[i]=as.integer(substring(strsplit(x," "), 12,13))
      }
     else{return(NA)}
    }else{return(NA)}

   }
 df = subset(df,select = -c(pickup_datetime))
 return(df)
}
##################################################end of data Preprocessing
 methods #######################################
train = data_cleaning_stage_cooridnates(train)
train = data_cleaning_stage_date_time(train)
train$fare_amount  = as.numeric(as.character(train$fare_amount))
train = train[train$fare_amount > 2.5 & train$fare_amount < 250,]
train = train[train$passenger_count > 0 & train$passenger_count < 7 ,]
test = data_cleaning_stage_cooridnates(test)
test = data_cleaning_stage_date_time(test)
test = test[complete.cases(test[ ,"passenger_count"]),]
test = test[test$passenger_count > 0 & test$passenger_count < 7 ,]
train=na.omit(train)
sum(is.na(train))
sort(train$fare_amount,decreasing = TRUE)
################################################## end of data cleaning
 ###################################################
library('sqldf')
library('ggplot2')

fareamount_by_year <- sqldf('select year, passenger_count , avg(fare_amount) as fare from
 train group by year,passenger_count')
ggplot(fareamount_by_year,aes(x=year, y=fare, color=passenger_count))+geom_point(data =
 fareamount_by_year, aes(group = passenger_count))+geom_line(data = fareamount_by_year,
 aes(group = passenger_count))+ggtitle("cab fare by year , passenger_count")

fareamount_by_month <- sqldf('select month, avg(fare_amount) as fare from train group by
 month')

fareamount_by_distance_year <- sqldf('select year, avg(fare_amount) as fare from train group
 by year, fare_amount')
ggplot(fareamount_by_distance_year,aes(x = year, y = fare, fill = year, label = year )) +
 geom_bar(stat = "identity",width = 0.20)+ggtitle(" avg cabfare amount vs avg distance by
 year")
################################################### end of data
 visualizations #######################################
MAPE = function(actual, prediction){return(mean(abs((actual - prediction)/actual))*100)}
RMSE=function(actual, predicted){ return(sqrt(mean((actual - predicted)**2))) }
```

```
MSE=function(actual, predicted){  return(mean((actual - predicted)**2)) }


error_metrics <- data.frame(matrix(ncol = 3, nrow = 3))
x <- c("DecisionTree", "RandomForest", "LinearRegression")
colnames(error_metrics) <- x
x <- c("MAPE", "MSE", "RMSE")
rownames(error_metrics)<-x
############################################ end of error metrics functions
 #######################################
train.index = createDataPartition(train$fare_amount, p = .20, list = FALSE)
train1 = train[ train.index,]
test1  = train[-train.index,]
rm(train.index)
fit = rpart(fare_amount ~ ., data = train1, method = "anova")
predictions_DT = round(as.numeric(predict(fit,test1[,-1])),2)
MAPE(test1[,1],predictions_DT)
MSE(test1[,1],predictions_DT)
RMSE(test1[,1],predictions_DT)
########################################## end of decision tree  mode
 ###########################################
lm_model = lm(fare_amount~.,data = train1)
vif(train1[,-1])
vifcor(train1[,-1],th = 0.9)
summary(lm_model)
predcition_Lr = predict(lm_model, test1[,2:10])
MAPE(test1[,1],predcition_Lr)
MSE(test1[,1],predcition_Lr)
RMSE(test1[,1],predcition_Lr)
############################################ end of linear regression
 #########################################
library('randomForest')
RF_model = randomForest(fare_amount ~ ., train1, importance = TRUE, ntree = 500)
RF_Predictions = predict(RF_model, test1[,-1])
round(MAPE(test1[,1],RF_Predictions),2)
round(MSE(test1[,1],RF_Predictions),2)
round(RMSE(test1[,1],RF_Predictions),2)
########################################## end of random forest
 ##############################################
test$fare_amount = round(predict(RF_model, test),2)
write.csv(test,"test_fare_amount_R.csv")
```

End of the R Code